# Réalisation de nos premiers widgets

version 1

Interface Homme-Machine : Unity

Voici les objectifs de ce sujet :

— Continuez à manipuler l'IDE `Unity`.

— Création d'un *widget* complexe.

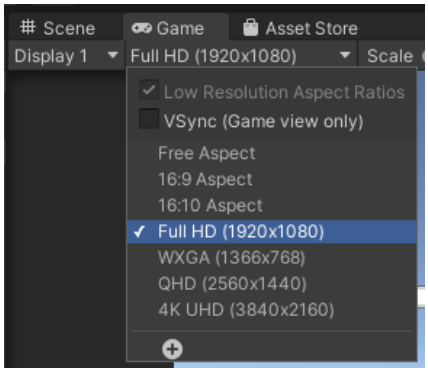
— Mécanisme des préfabriqués.

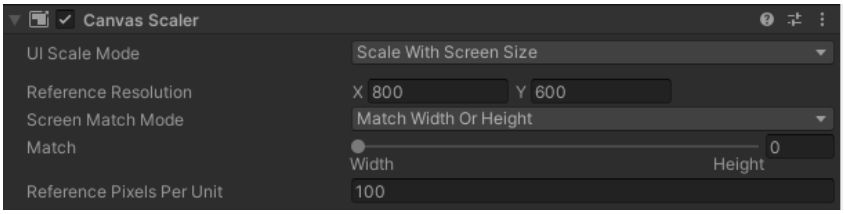
— Exportation de son travail : les *unity package*.

## Description générale du TP

La fois précédente, nous avons réalisé notre premier projet Unity et nous avons vu la manipulation de l'interface. La manipulation est simple (mais nécessite quelques calibration). Nous avons réalisé entre autre notre première interface très simplifiée, reposant sur la mécanique des ancres (voire aucune pour certains d'entre vous). La mécanique des ancres n'est pas aisé surtout lorsque les ratio de l'écran peuvent changer (réalisation d'une application PC et une application Android). Ce dernier point ne sera pas étudié dans ce TP car cela rentre dans les aspects très avancées. Cependant, pour le lecteur volontaire ou si vous finissez le TP (donc pas au début) il serait intéressant de regarder ce pointeur `https://docs.unity3d.com/2020.1/Documentation/Manual/HOWTO-UIMultiResolution.html`.

Quoi qu'il arrive, on vous demande dans un premier temps de créer un NOUVEAU projet vide et de fixer une résolution raisonnable [1] de votre Canvas, dans le menu Game de la fenêtre (sinon il s'agit de paramètre spécifique au déploiement de votre application). Modifier aussi en sélectionnant dans le Canvas de la hiérarchie, le composant gérant la mise à l'échelle en fonction de la résolution (`Canvas Scaler`) et respectant les paramètres à la résolution prêt l'image de droite.

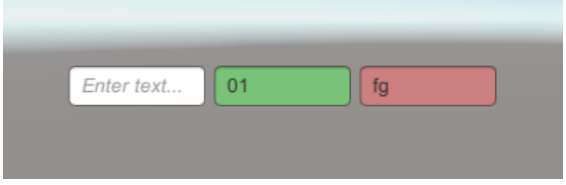



Une fois réalisez ça, nous pouvons passer à la réalisation de widget complexe. Pour expliquer ce terme, nous allons réalisé un agglomérat de widget existant avec un ou plusieurs scripts pour régir le comportement global du widget.

## 1 Premier widget complexe : FormattedInputField

L'objectif de ce widget est de réaliser une zone de texte qui change de couleur selon une expression régulière particulière comme sur l'image ci-dessous, où nous avons 3 `FormattedInputField`, ainsi lorsque la saisie est vide (le widget de gauche), nous avons une couleur standard, la présence d'un nombre engendre une couleur (le widget au centre) et une autre couleur si l'expression régulière n'est pas présente (le widget à droite).




Pour cela, nous vous donnons le code suivant qui vient d'un programme C# basique en dehors d'Unity :

1. La résolution que j'ai dans mes versions sont par défaut le `1024x768`.

```
Console.WriteLine("Regex␣experimentation");
string regex = "[0-9]+";
string montext = "bonjour";

if (System.Text.RegularExpressions.Regex.IsMatch(montext, regex))
       Console.WriteLine("Le␣text␣match␣la␣regex");
else
       Console.WriteLine("le␣text␣ne␣match␣pas␣la␣regex");
```

D'un point de vue conceptuel, nous devons faire les étapes suivantes (si vous faite autrement tant pis pour vous, même si cela peut marcher) :

1. Créez les attributs publique correspondant aux couleurs et les initialisées directement.
2. Créez l'attribut de la regex sous forme de chaine de caractère.
3. Réalisez la callback lorsqu'on change le texte dans le champ de saisie.
4. La couleur qu'il faut changer est celle du composant image.

**Information**

Pour réaliser le dernier point, il faut bien se rappeler de la séance précédente et de la partie sur l'inspecteur des objets en Unity. En effet, si on regarde bien nous avons différents composants pour le champs de saisie. Le composant qui nous intéresse est l'image pour changer l'attribut color pour répondre à nos conditions. Pour cela, il nous suffit à partir du `gameObject` de récupérer l'objet souhaité :

```
Image image = gameObject.GetComponent<Image>();
```

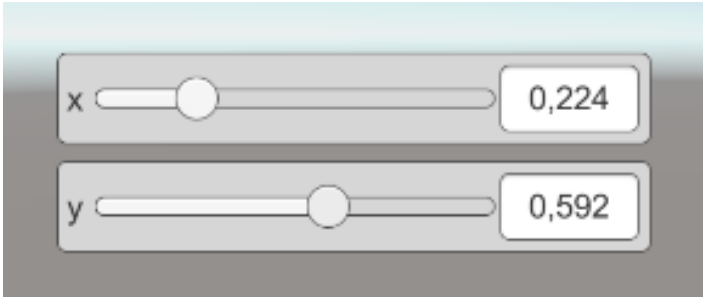
J'insiste que `GetComponent` récupère le premier composant du type souhaité dans le gameObject courant, il existe une version permettant de récupérer tous les composants d'un type cible.

```
Image[] images = gameObject.GetComponents<Image>();
```

Réalisez le widget voulu, en attachant une importance à la hiérarchie dans la structure de votre projet et aux noms que vous adoptez (harmonisation et uniformisation).

## 2 Widget : ComplexSlider

Dans le précédent widget nous avions un unique widget et ici nous allons composé un widget complexe à partir de 3 widgets basiques. Pour cela nous allons réaliser des curseurs complexes en composant un texte qui servira de label, d'un slider basique et d'un champ de saisie sur les nombres pour visualiser la valeur du slider et/ou le modifier manuellement. Ci-dessous, vous avez une image qui illustre 2 ComplexSlider pour choisir des coordonnées (x,y).




**Information**

Généralement un widget complexe doit avoir un panel à la base pour servir de fond visuel que vous opacifierez par défaut et qui permet de hiérarchiser les 3 sous-widgets le composant.

Vous devez réaliser un tel composant et une fois la partie réalisez, vous réaliserez un unique script qui gouverne tous les aspects comportemental. En particulier, lorsqu'une valeur est modifié, cela impacte l'autre widget pour avoir toujours une cohérence entre le champ de saisie et le slider basique.

Pour cela, nous allons exploiter la hiérarchie de notre widget. Il existe deux mécaniques pour retrouver les sous-fils :

1. via les fonctions de recherche basées sur leur nom.
2. via les mécaniques de recherche sur un type souhaité.

Réfléchissez sur les avantages et inconvénients des deux mécaniques en lisant la documentation associée à ces familles de fonction : `https://docs.unity3d.com/ScriptReference/GameObject.html`. En particulier on réfléchira que se passe-t-il si plusieurs objets sont du même type ? ou si l'utilisateur modifie le nom d'un sous-widget.

# 3 Exporter son travail

Bravo vous avez fait le plus gros essayer d'appeler l'enseignant pour qu'il vérifie votre développement ou critiquer vos noms et autres petits détails.

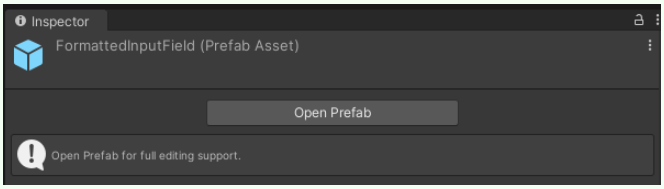
## 3.1 Préfabriqué

L'intéret de développer un widget est de pouvoir le réutiliser plusieurs fois sans devoir faire plusieurs manipulations identiques ou des `copier-coller`. Pour cela, Unity à la possibilité de faire des `Préfabs`, une sorte de sauvegarde de votre réalisation au sein d'un projet !

Pour ce faire il est assez simple lorsque vous avez fini un widget qui n'a pas de dépendance extérieur, c'est à dire que le widget est autonome (sinon ils risquent d'être sauvegardé), il suffit de glisser le gameObject de la hiérarchie à la zone des Assets. Ainsi l'icone du gameobject devient bleu ! C'est ainsi qu'on sait qu'il s'agit d'un Préfab.

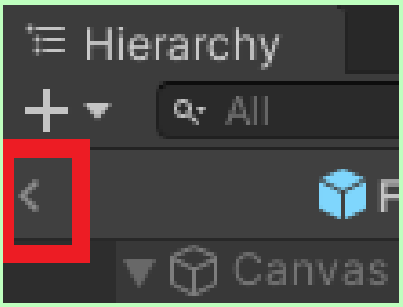
**Information**

Lorsque vous sélectionnez un Préfab dans la zone des Assets vous avez un nouveau bouton dans l'inspecteur qui vous permet de modifier le préfab.




**Attention**

Quand vous modifiez un Préfab vous êtes dans un environnement particulier, il est essentiel de revenir au plus vite dans le mode standard en particulier pour sauvegarder et ainsi éviter des petits problèmes.
Pour quitter, il faut cliquer sur le bouton retour (noté par $'<'$) au niveau de la hiérarchie.



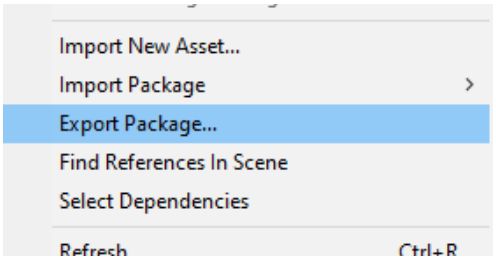

Réalisez des préfabs de vos widgets et les ayant nettoyés si besoin AVANT ! Ensuite instancié pour expérimenter deux ou trois de votre préfab. Pour cela, il suffit de glisser votre préfab depuis l'Assert vers la vue 3D ou la hiérarchie à l'endroit souhaité.

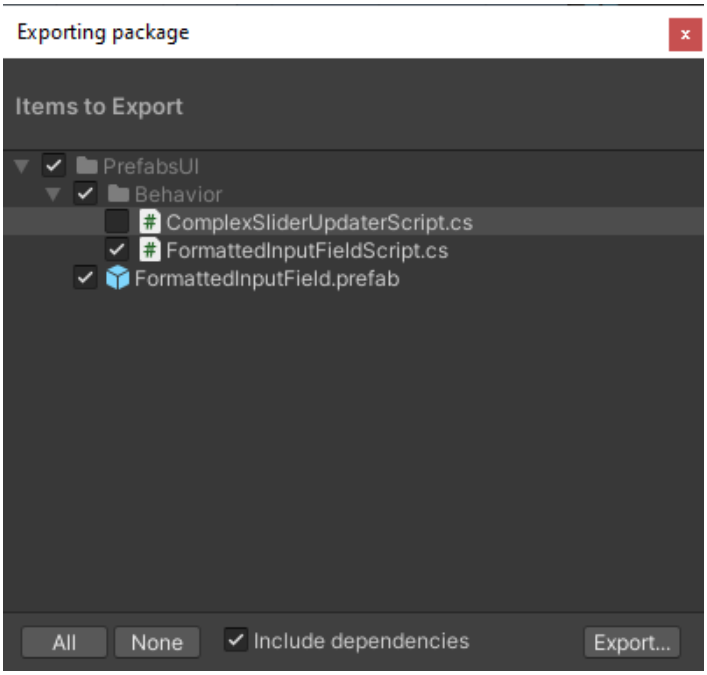
### 3.2 Export vers d'autre projet Unity

### 3.3 Export

Maintenant que vous avez la possibilité de sauvegarder et réutiliser votre production, au sein d'un projet, il est fréquent de devoir les exporter pour les utiliser dans d'autre projet. Pour cela, il faut exporter des Préfabs dont vous connaissez les dépendances, en sélectionnant votre Préfab dans les Assets et en faisant un clique droit dessus (attention on vous demande de tester l'export d'un widget dans un premier temps, puis plusieurs widgets dans un second temps afin que vous compreniez les dépendances). Dans le menu contextuel, sélectionnez `'Export Package...'`.

Dans la nouvelle fenêtre vous devez sélectionner les bonnes dépendances de votre widget. En particulier, par défaut Unity sélectionne tous les scripts sans distinction car il n'arrive pas à calculer les dépendances correctement. Vous devez donc sélectionner les bonnes dépendances de scripts manuellement. Puis cliquez, sur le bouton export pour sauvegarder le package unity.




### 3.4 Import

Pour tester nos paquets unity créés à l'étape précédent, c'est très simple ! Il suffit de créer un nouveau projet Unity et glissez votre fichier `.unitypackage` dans les Assets et de se laisser guider par le menu.

**Information**

Une alternative consiste juste à faire un clique droit dans la zone des Assets et cliquer sur l'import d'un paquet personnalisé.

Votre widget est prêt à l'emploi. La mécanique des unitypackage est l'une des façons de se partager le travail lorsqu'on est plusieurs sur un même projet et que cela s'y prête bien.

## 4 Toujours plus loin

Reprenez votre TP précédent ou un nouveau projet qui exploite vos nouveaux widgets, histoire de voir une application complète et comprendre que votre prefab doit bien être autonome pour embarquer uniquement son comportement intrinsèque !